Open Problems in FIFO Scheduling with Multiple Offsets

Mitra Nasri[1†], Robert I. Davis[2], and Björn B. Brandenburg[1]
[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] University of York, UK, and Inria, France

I. Introduction

First-In-First-Out (FIFO) is a widely used scheduling policy that can easily be implemented in hardware or software. With FIFO scheduling, the order in which jobs of tasks are executed depends only on their release times. This means that there are no preemptions, since no future job can have a higher priority than one that has already been released. Since tasks are not preempted, their *worst-case execution times* (WCETs) can be estimated with a higher degree of accuracy [1]. Further, on a uniprocessor, non-preemptive execution means that a running task has exclusive access to shared resources. This allows FIFO scheduling to reduce both design complexity and implementation overheads [2]. FIFO scheduling is also *sustainable* w.r.t. a reduction in execution times, i.e., if a task set is schedulable under the FIFO policy when all jobs exhibit their WCETs, then the system remains schedulable even if some jobs require less execution time [3]. As a result, in the absence of release jitter, it is possible to use a *simulation-based* schedulability test to obtain exact response-time analysis for periodic tasks under FIFO scheduling.

The key issue with FIFO scheduling is that it does not take task deadlines into account and so is ineffective at meeting time constraints [3]. This is because once a task enters the FIFO queue, its position in the queue cannot be changed even if a more urgent task is released later. FIFO scheduling of a hard real-time system therefore usually implies a severe under-utilization of the processing resource [2]. In an attempt to improve schedulability, Altmeyer et al. [3] proposed a method where offsets are chosen randomly until an assignment is found that is schedulable using FIFO scheduling; however, this solution does not scale well beyond a few tasks or to task sets with a high utilization [2]. We note that this is an example of the *classic* offset assignment problem, where the goal is to find an offset for each task such that the task set becomes schedulable by the given scheduling algorithm. In this case, the initial offset chosen for each task affects both the release times and the absolute deadlines of its jobs.

Recently, Nasri et al. [2] introduced a novel and different way of using offsets. Here, the initial offsets are assumed to be fixed, and hence the absolute deadlines for the jobs are also fixed. With the approach of Nasri et al. [2], jobs are partitioned into different groups and each group has a different additional relative offset applied. These relative offsets affect only the release times of the jobs, but not their deadlines. Fig. 1-(a) shows a FIFO schedule using multiple offsets. Note that this task set is not schedulable by any work-conserving policy if each task has only one offset (see Fig. 1-(b)). This example illustrates that going beyond one offset per task can yield significant gains in schedulability [2].

Next, we precisely define the relative offset assignment problem with multiple offsets, as explored by Nasri et al. [2]. We assume that a periodic task set is represented by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task τ_i is characterized by its period T_i, deadline D_i, WCET C_i, and initial offset O_i. In order to express multiple offsets for a task, we use offset pairs denoted by $OP = (k, o)$, where k is the job index (from the start of the hyperperiod) from which a new offset o is applied. For example, an offset pair $(3, 12)$ means that starting from the third job in the hyperperiod, each job of the task has an offset of $12 + O_i$, until this is changed by a later offset pair (if any). More precisely, the release time and deadline of the j^{th} job of task τ_i with relative offset $o_{i,x}$ are $r_{i,j} = O_i + (j-1) \cdot T_i + o_{i,x}$ and $D_{i,j} = O_i + (j-1) \cdot T_i + D_i$ respectively. (Note that the relative offset does not alter the deadline).

II. Challenges and Open Problems

Open Problem 1: Given a periodic task set τ, for each task τ_i (characterized by C_i, T_i, D_i, O_i) find a set of offset pairs $\hat{O}_i = \langle (k_{i,1}, o_{i,1}), \ldots, (k_{i,m_i}, o_{i,m_i}) \rangle$ such that the resulting task set is schedulable using FIFO scheduling.

Open Problem 1 comes with several challenges. Firstly, this problem is strongly NP-Hard since any general solution could also be used to solve the non-preemptive scheduling problem for periodic tasks, which is known to be strongly NP-hard [4]. Secondly, iterative approaches that attempt to find offsets for tasks one after another cannot easily be applied since changing the offset of one task may change the alignment of the releases of that task w.r.t. all other tasks, resulting in a totally different and potentially infeasible schedule [2]. This issue is more important for non-preemptive scheduling, since due to the blocking effect a change in the schedule of a lower-priority task affects both higher and lower priority tasks as well as the next job of the task itself. This makes the offset-assignment problem more difficult than the commonly studied preemptive case under fixed-priority scheduling. Interestingly, in the case of sets of periodic tasks with non-harmonic periods, it may not be possible

[†] The first author is supported by a fellowship from the Alexander von Humboldt Foundation.

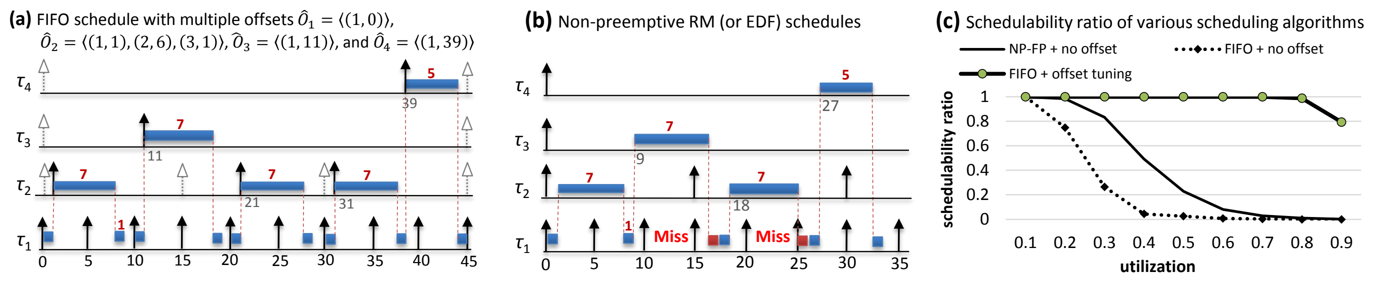

Fig. 1. **(a)** shows a FIFO schedule with multiple offsets, **(b)** shows a non-preemptive rate-monotonic schedule for the same task set, and **(c)** shows a comparison between the schedulability ratio of various algorithms according to the results of Nasri et al. [2] for task sets similar to those used in automotive systems.

to find a single offset per task that ensures schedulability. Fig. 1-(b) shows an example task set that can be scheduled with two offsets for a task, but is infeasible with only one.

An extreme solution would be to assign one offset to each job of a task. In that case, the problem reduces to one of finding a feasible non-preemptive schedule, since the offset of each job can point to its start time in the non-preemptive schedule. This approach, however, requires a large amount of memory to store the offset pairs, which leads to the following open problem.

Open Problem 2: Solve Open Problem 1 such that the total number of offset pairs is minimized, i.e., $\min \sum_{i=1}^{n} |\hat{O}_i|$.

Recently, we have proposed a heuristic for solving Open Problem 1 [2], which is based on a reverse method, i.e., instead of proposing an algorithm to find the offset pairs while *building the schedule*, our method starts from a given feasible *reference schedule* and tries to assign offset pairs such that the FIFO scheduler mimics the given reference schedule. This technique, which is called *offset tuning*, finds a small set of offsets such that the resulting FIFO schedule becomes *equivalent* to the reference schedule, i.e., the resulting FIFO schedule has the same *job ordering* as the reference schedule and no job in the FIFO schedule finishes later than the corresponding job in the reference schedule. Thus, offset tuning guarantees FIFO schedulability by design, provided that the reference schedule is feasible. Offset tuning has $O(M \log M)$ computational complexity, where M is the total number of jobs in the hyperperiod. In order to obtain high levels of schedulability, we used the non-preemptive, non-work-conserving *Critical-Window EDF* (CW-EDF) [5] scheduling algorithm to produce the reference schedule. This combination enables high schedulability to be achieved while retaining the low runtime of FIFO scheduling with only a small added memory footprint [2]. Fig. 1-(c) compares the schedulability ratio of FIFO, *non-preemptive fixed-priority* (NP-FP) with *rate-monotonic* priorities, and FIFO with offset tuning (FIFO-OT). It is worth noting that the offset tuning method [2] greedily minimizes the number of offsets used for each individual task; however, this does not necessarily result in the minimum number for the task set as a whole. The proposed solution thus does not optimally solve Open Problem 2, even though empirically it tends to perform well [2]. Further, the performance of the offset-tuning heuristic for other types of reference schedules (e.g., obtained from an ILP solver) is unclear and may be much less efficient.

One possible extension is to consider systems in which tasks exhibit release jitter. Release jitter, which can arise due to timer interrupts, interrupt latency, network delays, etc., can potentially change the order of jobs in the FIFO queue at runtime. Such runtime unpredictability renders Open Problem 1 much more challenging, since the relative offsets used must make the resulting FIFO schedule consistent (or at least schedulable) for any combination of release jitter for all of the different tasks.

Open Problem 3: Given a periodic task set τ, for each task τ_i (characterized by C_i, T_i, D_i, O_i, J_i) find a set of offset pairs $\hat{O}_i = \langle(k_{i,1}, o_{i,1}), \ldots, (k_{i,m_i}, o_{i,m_i})\rangle$ such that the resulting task set is schedulable using FIFO scheduling.

Another possible extension is to relax the constraints that the initial offsets O_i are fixed. This leads to an open problem that builds upon the classic offset assignment problem.

Open Problem 4: Given a periodic task set τ, for each task τ_i (characterized by C_i, T_i, D_i) find an initial offset O_i and a set of offset pairs $\hat{O}_i = \langle(k_{i,1}, o_{i,1}), \ldots, (k_{i,m_i}, o_{i,m_i})\rangle$ such that the resulting task set is schedulable using FIFO scheduling.

References

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, 2008.
[2] M. Nasri, R. I. Davis, and B. Brandenburg, "FIFO with Offsets: High Schedulability with Low Overheads," in *RTAS*, 2018, pp. 271–282.
[3] S. Altmeyer, S. Sundharam, and N. Navet, "The case for FIFO real-time scheduling," University of Luxembourg, Tech. Rep., 2016.
[4] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *RTSS*, 1991, pp. 129–139.
[5] M. Nasri and G. Fohler, "Non-work-conserving non-preemptive scheduling: motivations, challenges, and potential solutions," in *ECRTS*, 2016, pp. 165–175.